Docker for Robotics

How to use and create Docker containers

Presenter:
Corrie Van Sice
Research Scientist Associate
Texas Robotics, Nuclear and Applied Robotics Group

Join Zoom Meeting
https://utexas.zoom.us/j/92548905256
Meeting ID: 925 4890 5256

Workshop Resources

Go to the Github repository:

https://github.com/ut-texas-robotics/docker_workshop

- This Presentation
- Exercise Files and Docs
- Code

Format

1. **Presentation**
    - Introduce Core Concepts
2. **Workshop Exercises**
    - Install Docker Engine and Explore the CLI
    - Create a Custom Docker Image
    - Configure with Docker Compose
3. **Demo**
    - Robot ROS Demo
4. **Presentation & Discussion**
    - Development Strategies

Learning Goals

- What is Docker Engine and the CLI
- Using images and containers
- Write Dockerfiles to create images
- Using docker compose to build and launch projects
- Understanding volumes, devices and network
- Interfacing with robot hardware and ROS
- Development strategies using git and scripting

What is Docker?

Docker is a tool that allows you to package applications and their dependencies into lightweight containers. These containers run consistently across various environments.

Important Components:
Images, Containers, Daemon, CLI, Registry

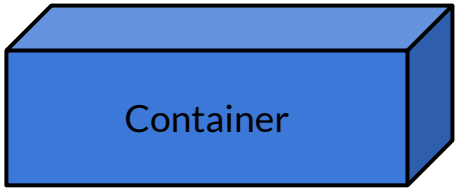
Docker Image

A Docker image is a **read-only** snapshot of a filesystem, comprising libraries, environment variables, and configurations needed to run an application's code. The application code may or may not be included in the image. Images are stored in layers and are highly optimized for performance and resource efficiency. Images are often created from other images. They can be downloaded from a registry, created from Dockerfiles or created from container states.

Docker Container

A container is a **runnable and read/writable** instance of a docker image. It is defined by an image as well as configuration options you provide when you create or start it. A container is a highly isolated environment–you can run multiple containers simultaneously; connect containers to volumes, networks, and other subsystems on the host, as well as each other. **When a container is removed, any changes to its state that aren't stored in persistent storage disappear.**

Image vs Container



Read only, immutable

Describes a container

Composed of layers

Contains meta-data

Read/Write, Changes state

Isolated environment

Can connect to volumes, networks, devices, etc.

Start, stop, remove

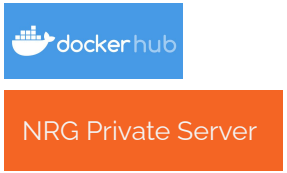
Where do images come from?

Docker Registries

A place where images are stored online.

Can be private or public.

Images are already built.



Dockerfiles

Files used to build new images.

Often included in git repos, rather than a link to a registry.

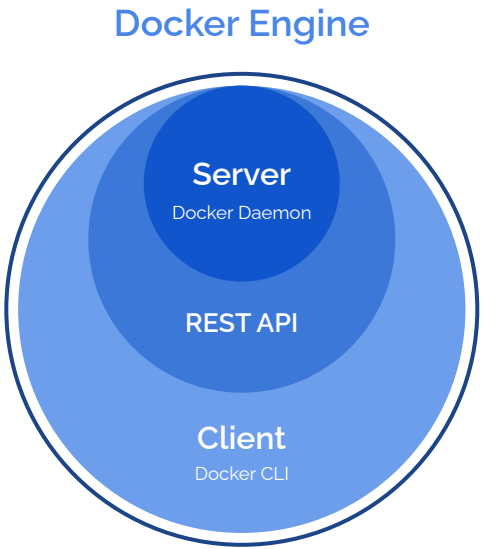
Image must be built.

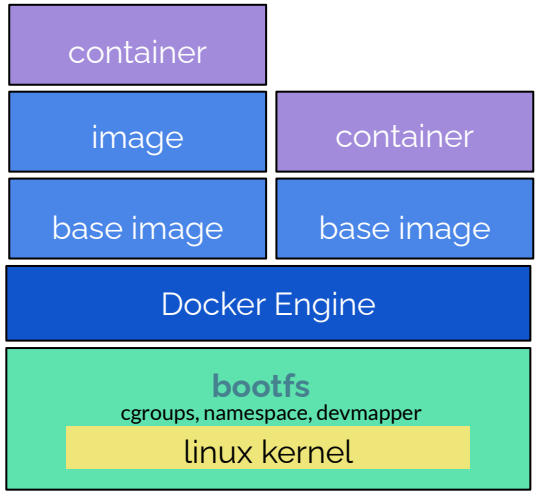
```
FROM nvidia/cuda:11.8.0-cudnn8-devel-ubuntu18.04

SHELL ["/bin/bash", "-c"]
RUN apt-get update &&\
        apt-get -y install nano tmux curl
WORKING_DIR /root
RUN git clone –recursive
        https://github.com/ut-texas-robotics/some_repo.git &&\
        cd some_repo &&\
        make
CMD [“/bin/bash”]
```

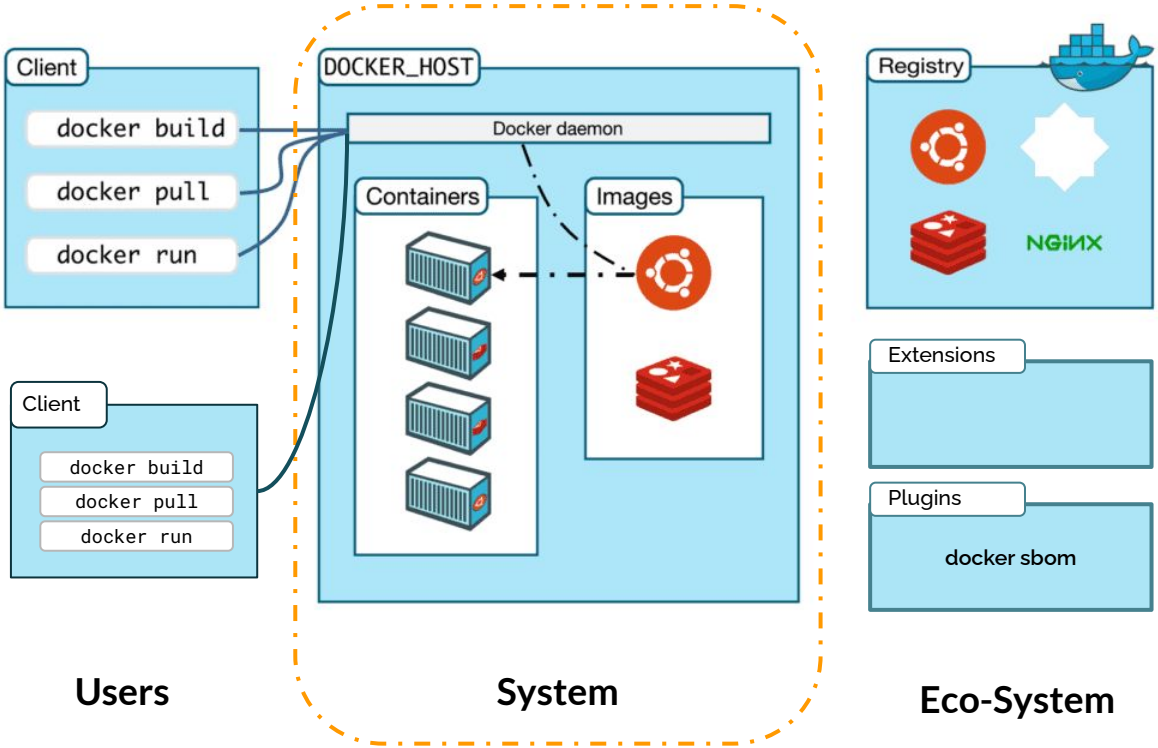
Docker Engine

Client/Server model: the server is exposed via a REST API, which clients use to make requests of the server. It interacts with a Linux Kernel host system via cgroups, namespaces, and device mapper. When you run a container, Docker creates a set of namespaces for that container.





Docker Architecture



Hands-On

Install Docker Engine and Explore the CLI

Install Docker Engine

- Install Docker Engine on Ubuntu
- Install NVIDIA Container Toolkit
- Post-Installation Steps
- Software Bill of Materials plugin

Docker Commands

`docker pull`	pull an image from a registry
`docker sbom`	list the “software bill of materials” of an image
`docker image ls`	list the images on your system
`docker ps` `docker container ls`	list the running containers (user `-a` to list all containers)
`docker run`	create a container from an image
`docker exec`	execute a command in a running container
`docker stop`	stop a container
`docker container rm`	delete a container

Dockerfiles

Create new images

Docker images are essentially snapshots of a file system.

Dockerfiles are scripts used to build custom Docker images.

When building images, it's important to grasp the concept of layers and the build structure created by commands.

Images for Development

```
> Docker
    Dockerfile
    docker-compose.yml
    entrypoint.sh
    bash_utils.sh
    robot.env
```

Q: What goes in my Dockerfile?

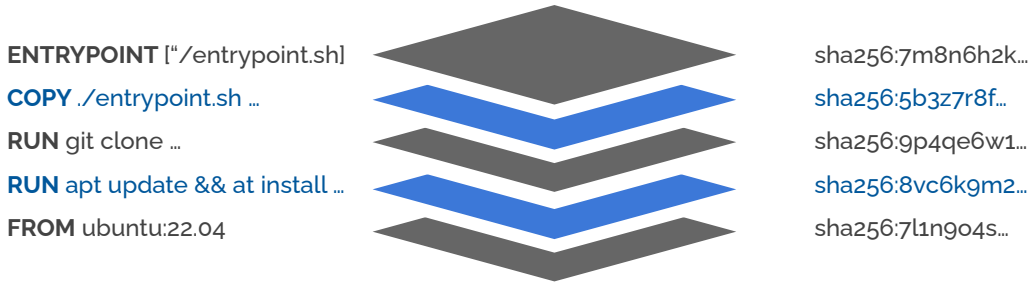
A:

When writing an image for development environments, it can be helpful to consider the function of an image as building that environment, and not packaging an entire application or configuring all parameters. You can use other tools for configuration.

Many docker users write images for containing entire applications and setting configs, and this is not wrong. It just depends on how the container is intended to be used.

Layers

A Docker image is composed of multiple layers. Each layer represents a set of changes to the file system. This layering system allows for efficiency because when an image is modified, only the affected layers need to be updated, not the entire image. This makes image creation and distribution faster and more efficient.



Dockerfile

```
FROM ubuntu:22.04

SHELL ["/bin/bash", "-c"]
RUN apt-get update &&\
        apt-get -y install nano tmux curl
WORKING_DIR /root
RUN git clone –recursive
        https://github.com/ut-texas-robotics/some_repo.git &&\
        cd some_repo &&\
        make
CMD [“/bin/bash”]
```

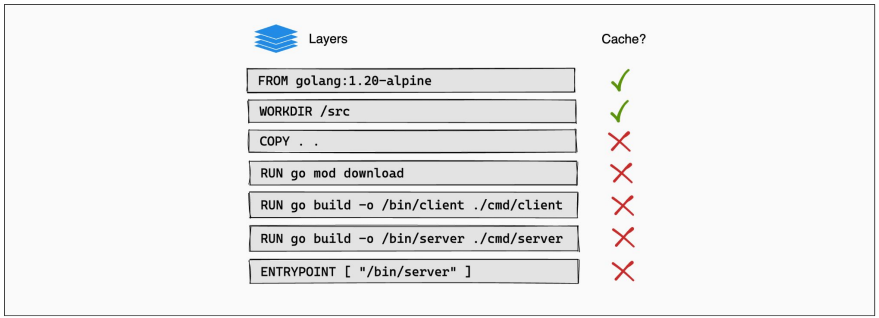
History

```
IMAGE          CREATED         CREATED BY                                      SIZE      COMMENT
66e8b66616e2   3 days ago      ENTRYPOINT ["/bin/bash" "-l" "-c" "/entrypoi…   0B        buildkit.dockerfile.v0
<missing>      3 days ago      COPY ./entrypoint.sh /entrypoint.sh # buildk…   1.63kB    buildkit.dockerfile.v0
<missing>      3 days ago      RUN /bin/bash -l -c echo "" # buildkit          0B        buildkit.dockerfile.v0
<missing>      3 days ago      SHELL [/bin/bash -l -c]                         0B        buildkit.dockerfile.v0
<missing>      3 days ago      RUN /bin/bash -c tar -xzvf cmake-3.27.7-linu…   147MB     buildkit.dockerfile.v0
<missing>      4 days ago      RUN /bin/bash -c wget https://github.com/Kit…   51.7MB    buildkit.dockerfile.v0
<missing>      4 days ago      RUN /bin/bash -c cd c-blosc &&    mkdir bui…   17.5MB    buildkit.dockerfile.v0
<missing>      4 days ago      RUN /bin/bash -c git clone https://github.co…   18.3MB    buildkit.dockerfile.v0
<missing>      4 days ago      WORKDIR /root                                   0B        buildkit.dockerfile.v0
<missing>      4 days ago      RUN /bin/bash -c apt update && apt upgrade -…   2.47GB    buildkit.dockerfile.v0
<missing>      4 days ago      SHELL [/bin/bash -c]                            0B        buildkit.dockerfile.v0
<missing>      3 weeks ago   /bin/sh -c #(nop)  CMD ["/bin/bash"]              0B
<missing>      3 weeks ago   /bin/sh -c #(nop) ADD file:63d5ab3ef0aab308c…   77.8MB
<missing>      3 weeks ago   /bin/sh -c #(nop)  LABEL org.opencontainers.…   0B
<missing>      3 weeks ago   /bin/sh -c #(nop)  LABEL org.opencontainers.…   0B
<missing>      3 weeks ago   /bin/sh -c #(nop)  ARG LAUNCHPAD_BUILD_ARCH     0B
<missing>      3 weeks ago   /bin/sh -c #(nop)  ARG RELEASE                  0B
```

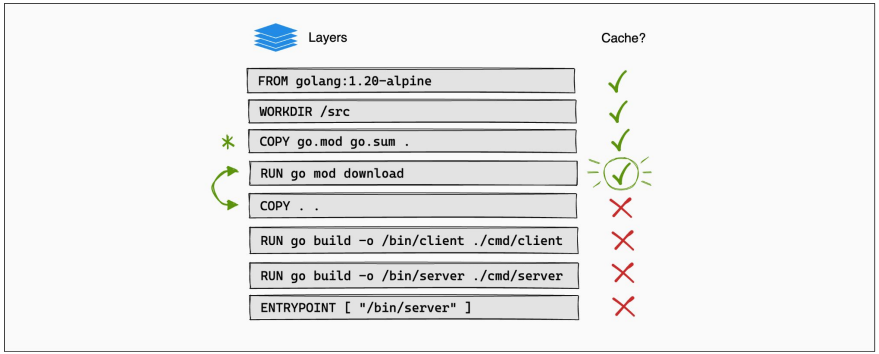
Cached layers

When you run a build, the builder attempts to reuse layers from earlier builds. If a layer of an image is unchanged, then the builder picks it up from the build cache. If a layer has changed since the last build, that layer, and all layers that follow, must be rebuilt.

changed files rebuild from where the change originated



organizing commands can save build time



Dockerfile Commands

FROM
- pull a base image from a registry, docker hub, or your own collection
- examples: ros, conda, ubuntu:22.04, etc.

RUN
- each of these commands is a layer
- RUN is not the same as doing an individual bash command
- RUN <command> or RUN ["executable", "param1", "param2"]

ENV
- these are environment variables you need in the build process, not necessarily in your config

SHELL
- set the default shell to use - if left undefined, its /bin/sh

COPY
- copy a file into the container, such as an entrypoint script

CMD
- CMD is not the same as RUN
- this is the command that is executed when you start a container.
- this does not contribute to building your image, it only provides a directive for what should be executed when you start a container from the image
- CMD ["executable","param1","param2"], or CMD <command>, or CMD ["param1", "param2"]

ENTRYPOINT
- this is a script that runs when you start a container, rather than a simple command
- it is copied into the image when you build it
- the benefit is that you can dynamically make changes to the container using the script
- the challenge is that you may need to pass parameters to it

Docker Compose

Configuring containers with a docker-compose.yaml

Setup the image build and the container configs.

Run multiple containers at once, such as a database and an app.

Configure volumes, devices, runtime, and more.

Robot Demo

Running a BWIbot from Docker

Start a bot from a container.

https://github.com/utexas-bwi/bwi-docker

Dev Methods

Using a container as a development environment.

Edit code on the host.

Run git commands on the host.

Build/run in the container.

Levelling up.